Бесплатная электронная книга

УЧУСЬ

mvvm

Free unaffiliated eBook created from
Stack Overflow contributors.

#mvvm

1

1: mvvm 2

2

Examples 2

C # MVVM 2

9

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: mvvm

It is an unofficial and free mvvm ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official mvvm.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с mvvm

замечания

В этом разделе представлен обзор того, что такое mvvm, и почему разработчик может захотеть его использовать.

Следует также упомянуть о любых крупных предметах в mvvm и ссылаться на связанные темы. Поскольку Documentation for mvvm является новым, вам может потребоваться создать начальные версии этих связанных тем.

Examples

C # MVVM Резюме и полный пример

Резюме:

MVVM - это архитектурный шаблон, который представлен тремя различными компонентами: **Model** , **View** и **ViewModel** . Чтобы понять эти три уровня, необходимо кратко определить их, а затем объяснить, как они работают вместе.

Модель - это слой, который управляет бизнес-логикой. Он извлекает и сохраняет информацию из любого источника данных для потребления **ViewModel** .

ViewModel - это слой, который действует как мост между **представлением** и **моделью** . Он может или не может преобразовывать необработанные данные из **Модели** в презентабельную форму для **представления** . Примерное преобразование: булевский флаг от модели до строки «True» или «False» для представления.

Вид - это слой, который представляет интерфейс программного обеспечения (то есть GUI). Его роль заключается в отображении информации *от* **ViewModel** к пользователю, и сообщить изменения информации обратно *в* **ViewModel.**

Эти три компонента работают вместе, ссылаясь друг на друга следующим образом:

- **View** ссылается на **ViewModel** .
- Модель **ViewModel** ссылается на **модель** .

Важно отметить, что **View** и **ViewModel** способны к двусторонней связи, известной как **привязка данных** .

Основным компонентом двусторонней связи (привязки данных) является интерфейс INotifyPropertyChanged .

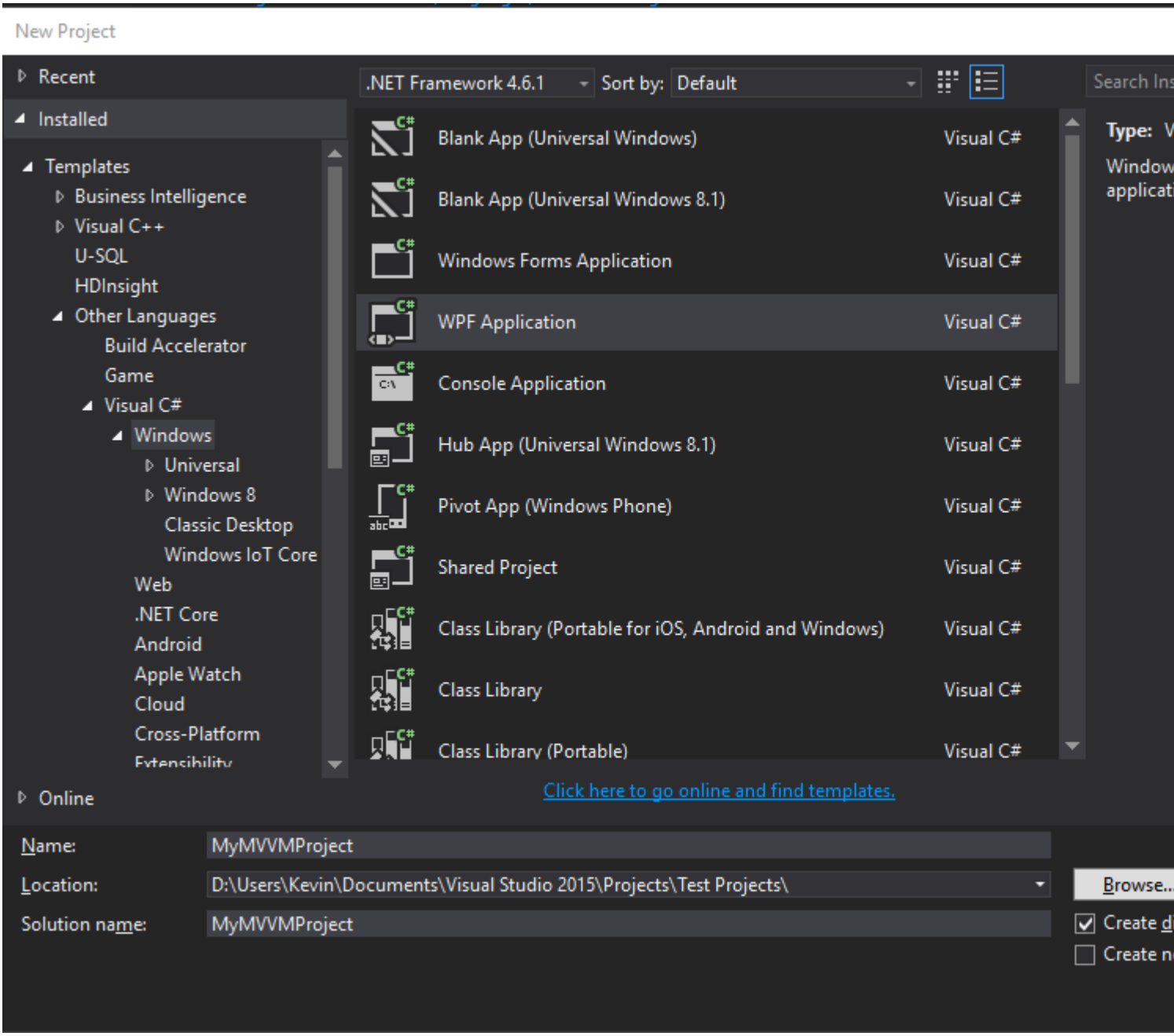
Используя этот механизм, **View** может изменять данные в **ViewModel** через пользовательский ввод, а **ViewModel** может обновлять **представление** с данными, которые могут быть обновлены с помощью процессов в **Модели** или с обновленными данными из репозитория.

Архитектура MVVM делает упор на **разделении проблем** для каждого из этих слоев. Разделение слоев приносит нам пользу:

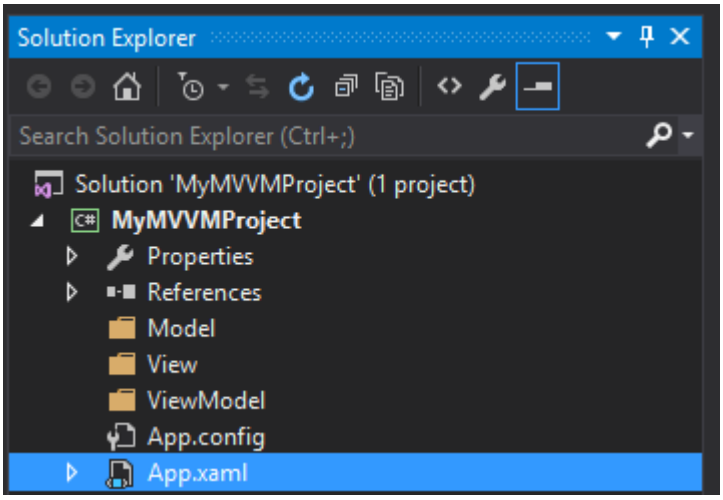
- **Модульность:** внутренняя реализация каждого уровня может быть изменена или заменена, не затрагивая других.
- **Повышенная способность к тестированию:** каждый уровень может быть **Unit Test** с поддельными данными, что невозможно, если код **ViewModel** написан в Code-Behind of **View** .

Сборка:

Создайте новый проект приложения WPF

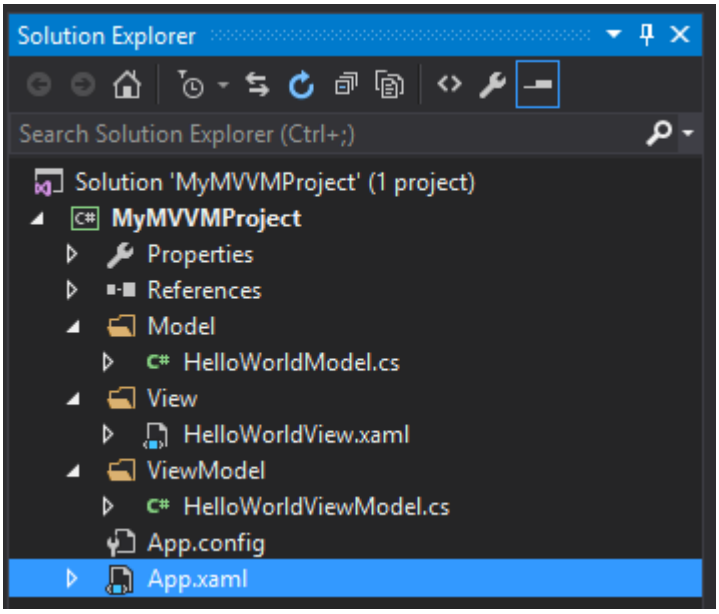


Создайте три новые папки в своем решении: **Model** , **ViewModel** и **View** и удалите исходный `MainWindow.xaml` , чтобы начать новый старт.



Создайте три новых элемента, каждый из которых соответствует отдельному слою:

- Щелкните правой кнопкой мыши папку « **Модель** » и добавьте элемент **класса** `HelloWorldModel.cs` .
- Щелкните правой кнопкой мыши папку **ViewModel** и добавьте элемент **класса** `HelloWorldViewModel.cs` .
- Щелкните правой кнопкой мыши папку « **Просмотр** » и добавьте элемент « **Окно» (WPF)** под названием `HelloWorldView.xaml` .



Alter `App.xaml` чтобы указать на новый **вид**

```
<Application x:Class="MyMVVMProject.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:MyMVVMProject"
             StartupUri="/View/HelloWorldView.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

ViewModel:

Начните сначала с создания **ViewModel** . Класс должен реализовать интерфейс `INotifyPropertyChanged` , объявить событие `PropertyChangedEventHandler` и создать метод для создания события (источник: MSDN: как реализовать уведомление об изменении свойств). Затем объявите поле и соответствующее свойство, чтобы вызвать метод `OnPropertyChanged()` в `OnPropertyChanged() set` свойств. Конструктор в приведенном ниже примере используется для демонстрации того, что **Модель** предоставляет данные **ViewModel** .

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
```

```
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;
using MyMVVMProject.Model;

namespace MyMVVMProject.ViewModel
{
    // Implements INotifyPropertyChanged interface to support bindings
    public class HelloWorldViewModel : INotifyPropertyChanged
    {
        private string helloString;

        public event PropertyChangedEventHandler PropertyChanged;

        public string HelloString
        {
            get
            {
                return helloString;
            }
            set
            {
                helloString = value;
                OnPropertyChanged();
            }
        }

        /// <summary>
        /// Raises OnPropertychangedEvent when property changes
        /// </summary>
        /// <param name="name">String representing the property name</param>
        protected void OnPropertyChanged([CallerMemberName] string name = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
        }

        public HelloWorldViewModel()
        {
            HelloWorldModel helloWorldModel = new HelloWorldModel();
            helloString = helloWorldModel.ImportantInfo;
        }
    }
}
```

Модель:

Затем создайте **модель** . Как указано ранее, **модель** предоставляет данные для **ViewModel** , вытаскивая их из репозитория (а также отбрасывая обратно в хранилище для сохранения). Это показано ниже с помощью метода `GetData()` , который возвращает простой `List<string>` . Бизнес-логика также применяется в этом слое и может быть замечена в методе `ConcatenateData()` . Этот метод строит предложение «Hello, world!» Из `List<string>` которое было ранее возвращено из нашего «репозитория».

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

namespace MyMVVMProject.Model
{
    public class HelloWorldModel
    {
        private List<string> repositoryData;
        public string ImportantInfo
        {
            get
            {
                return ConcatenateData(repositoryData);
            }
        }

        public HelloWorldModel()
        {
            repositoryData = GetData();
        }

        /// <summary>
        /// Simulates data retrieval from a repository
        /// </summary>
        /// <returns>List of strings</returns>
        private List<string> GetData()
        {
            repositoryData = new List<string>()
            {
                "Hello",
                "world"
            };
            return repositoryData;
        }

        /// <summary>
        /// Concatenate the information from the list into a fully formed sentence.
        /// </summary>
        /// <returns>A string</returns>
        private string ConcatenateData(List<string> dataList)
        {
            string importantInfo = dataList.ElementAt(0) + ", " + dataList.ElementAt(1) + "!";
            return importantInfo;
        }
    }
}
```

Посмотреть:

Наконец, **View** может быть построен. В этом примере ничего не нужно добавлять в код, хотя это может варьироваться в зависимости от потребностей приложения. Однако в XAML добавлено несколько строк. `Window` нуждается в ссылке на пространство имен `ViewModel` . Это сопоставляется с пространством имен XAML `xmlns:vm="clr-namespace:MyMVVMProject.ViewModel"` . Далее, Окно нуждается в `DataContext` . Для этого установлено значение `<vm:HelloWorldViewModel/>` . Теперь ярлык (или управление по вашему выбору) можно добавить в окно. Ключевым моментом на этом этапе является обеспечение того, чтобы вы **привязывали привязку** к свойству **ViewModel,** которое вы хотите отображать в качестве содержимого метки. В этом случае это `{Binding HelloString}` .

Важно привязать свойство, а не поле, так как в последнем случае **представление** не получит уведомление о том, что значение изменилось, поскольку метод `OnPropertyChanged()` будет только поднять `PropertyChangedEvent` на свойство, а не на поле.

```
<Window x:Class="MyMVVMProject.View.HelloWorldView"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MyMVVMProject.View"
        xmlns:vm="clr-namespace:MyMVVMProject.ViewModel"
        mc:Ignorable="d"
        Title="HelloWorldView" Height="300" Width="300">
    <Window.DataContext>
        <vm:HelloWorldViewModel/>
    </Window.DataContext>
    <Grid>
        <Label x:Name="label" FontSize="30" Content="{Binding HelloString}"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
    </Grid>
</Window>
```

Прочитайте Начало работы с mvvm онлайн: https://riptutorial.com/ru/mvvm/topic/4293/начало-работы-с-mvvm

кредиты

S. No	Главы	Contributors
1	Начало работы с mvvm	Community, Gabor Barat, Kcvin, Kevin Mills, Maverik, MotKohn, Umair Farooq